Discuss the problems of deadlock and starvation, and the different approaches to dealing with these problems. **(08 marks)**

Deadlock occurs when *each* transaction T in a set of *two or more transactions* is waiting for some item that is locked by some other transaction T' in the set

way to prevent deadlock is to use a **deadlock prevention protocol**.

One deadlock prevention protocol, which is used in conservative two-phase locking, requires that every transaction lock all the items it needs in if any of the items cannot be obtained, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items it needs. A second protocol, which also limits concurrency, involves *ordering all the items* in the database and making sure that a transaction that needs several items will lock them according to that order. This requires that the programmer is aware of the chosen order of the items, which is also not practical in the database context.

Another group of protocols that prevent deadlock do not require timestamps. These include the no waiting (NW) and cautious waiting (CW) algorithms.

In the **no waiting algorithm**, if a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not.

The **cautious waiting** algorithm was proposed to try to reduce the number of needless aborts/restarts. Suppose that transaction Ti tries to lock an item X but is not able to do so because X is locked by some other transaction Tj with a conflicting lock.

Another problem that may occur when we use locking is **starvation**, which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.

One solution for starvation is to have a fair waiting scheme, such as using a **first-come-first-served** queue; transactions are enabled to lock an item in the order in which they originally requested the lock. Another scheme allows some transactions to have priority over others but increases the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.

Explain Time stamp ordering algorithms? **(08marks)**

A schedule in which the transactions participate is serializable, and the *only equivalent* serial schedule permitted has the transactions in order of their timestamp values. This is called **Timestamp Ordering (TO)**

Basic TO Algorithm: The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in the following two cases:

1. Whenever a transaction T issues a write_item(X) operation, the following is checked:

a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some *younger* transaction with a timestamp greater than TS(T)—and hence *after* T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.

b. If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

2. Whenever a transaction T issues a read_item(X) operation, the following is checked:

a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T)—and hence *after* T in the timestamp ordering—has already written the value of item X before T had a chance to read X.

b. If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the *larger* of TS(T) and the current read_TS(X).

Strict Timestamp Ordering (TO):

Thomas's Write Rule: A modification of the basic TO algorithm, known as **Thomas's write rule**, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:

1. If read_TS(X) > TS(T), then abort and roll back T and reject the operation.

2. If write_TS(X) > TS(T), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of X. Thus, we must ignore the write_item(X) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).

3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write_item(X) operation of T and set write TS(X) to TS(T).

Explain why a transaction execution should be atomic? Explain ACID propertie (08 marks)

Transactions should posses the following (ACID) properties: Transactions should possess several properties. These are often called the **ACID properties,** and they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties:

1. **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
2. **Consistency preservation:** A transaction is consistency preserving if its complete execution take(s) the database from one consistent state to another.
3. **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **Durability or permanency:** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

The atomicity property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensure atomicity. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database.

What is two phase locking ? Describe with the help of an example (04 Marks)

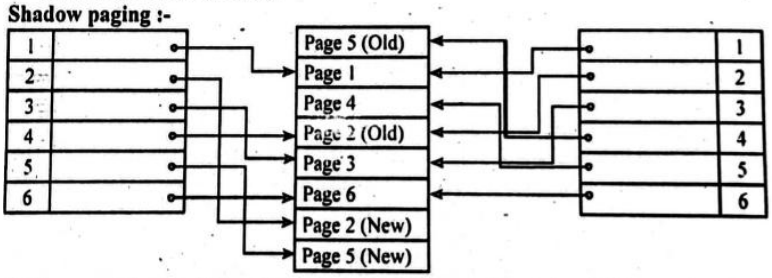
Two Phase Locking :- A transaction is said to follow the two-phase locking protocol if all locking operations precede the first unlock operation in the transaction.

Such a transaction can be divided in to two phases an expanding or growing phase during which new locks on items can be acquired but none can be released and a shrinking phase during which existing locks can be released but no new locks can be acquired

T_1	T_2
read - lock (y) :	read - lock (x) :
read - item (y0 :	read - item (x) :
write - lock(x) :	write - lock (y) :
unlock (y) ;	unlock (x)
read - item (x) :	read - item (y) :
x : = x +y ;	y ; = x + y :
Write - item (x) ;	Write - item (y) :
Unlock (x) :	Unlock (y) :

- Shadow paging considers the database to be made up of a number of fixed - size disk pages-say n for recovery purposes.
  - A directory with n entities is constructed , where the i^{th} entry points to the i^{th} database page on disk.
  - The directory is kept in main memory if it is not too large, and all references - reads or writes to database pages on disk go through it.
  - When a transaction begins executing, the current directory whose entries point to the most recent or current database pages on disk is copied into a shadow directory.
  - The shadow directory then saved on disk while the current directory is used by the transactions.
  - During transaction execution, the shadow directory is never modified . When a write-item operation is performed, a new copy of the modified database page is created , but the old copy of that page is not overwritten.
  - The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified disk block.
  - For pages updated by th transaction, two versions are directory and the new version by current directory.

Shadow paging :-



- To recover from a failure during transactions execution, it is sufficient to free the modified database pages and to discard the current directory.